Pythian Operational Visibility

Percona Live Santa Clara, 2015
April 13, 2015

Derek Downey
MySQL Principal Consultant

Laine Campbell
Co-Founder, Open Source Database Practice

Pythian®
love your data

ABOUT PYTHIAN

10,000

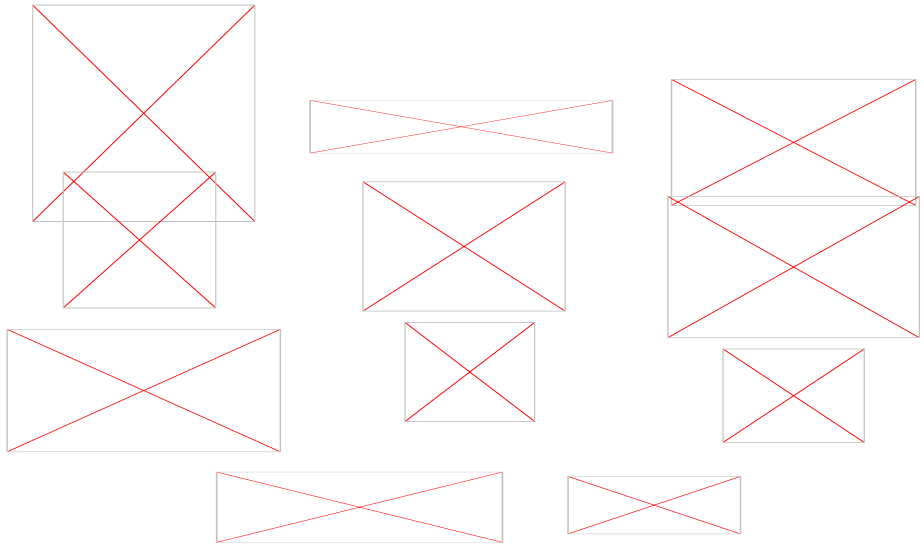
Pythian currently manages more than 10,000 systems.

350

Pythian currently employs more than 350 people in 25 countries worldwide.

1997

Pythian was founded in 1997



- 200+ leading brands trust us to keep their systems fast, relaible, and secure
- Elite DBA & SysAdmin workforce: 7 Oracle ACEs, 2 Oracle ACE Directors, 5 Microsoft MVPs, 1 Cloudera Champion of Big Data, 1 Datastax Platinum Administrator — More than any other company, regardless of head count
- Oracle, Microsoft, MySQL, Hadoop, Cassandra, MongoDB, and more.
- Infrastructure, Cloud, SRE, DevOps, and application expertise
- Big data practice includes architects, R&D, data scientists, and operations capabilities
- Zero lock-in, utility billing model, easily blended into existing teams.

content

- discussion
  - laine campbell
  - one hour
- set-up host environments to monitor
  - derek downey
  - 30 minutes
- review observability stack
  - laine campbell
  - 30 minutes
- attach to observability stack, hands-on, Q&A
  - derek downey and laine campbell
  - one hour

Our Goals: to understand

- observability objectives, principles and outcomes
- current state and problems
- metrics
- observability architecture
- choosing what to measure
  - business KPIs and ways to track them
  - pre-emptive and diagnostics measurements
- the Pythian opsviz stack
  - what it is
  - how to set it up
  - how to visualize data

operational visibility

continuous improvement

---

kaizen recognizes improvement can be small or large.

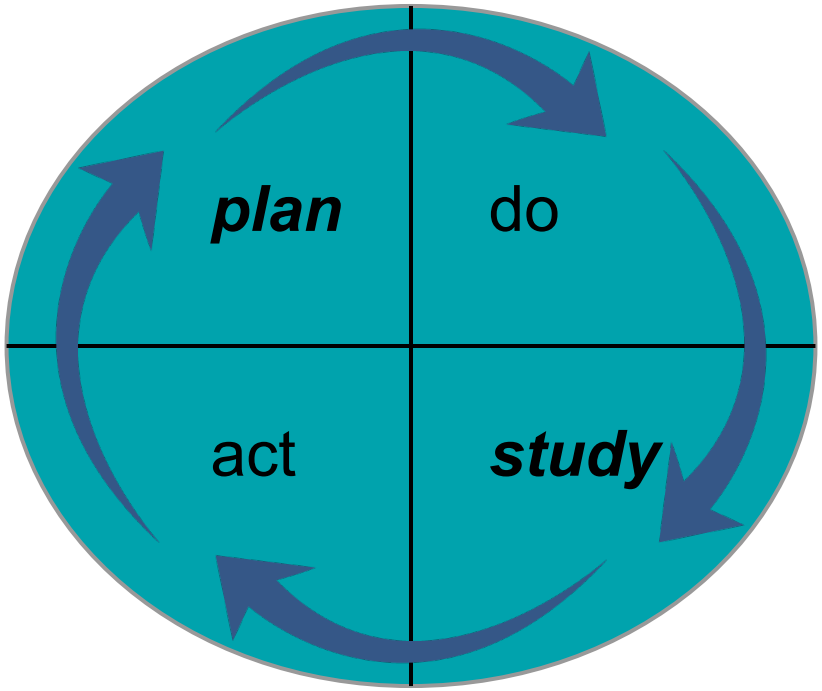
many small improvements can make a big change.

Pythian®
love your data

to improve a system, you must...

- understand it
- describe it
- involve, and motivate all stakeholders

enabling kaizen



we can do none of this...

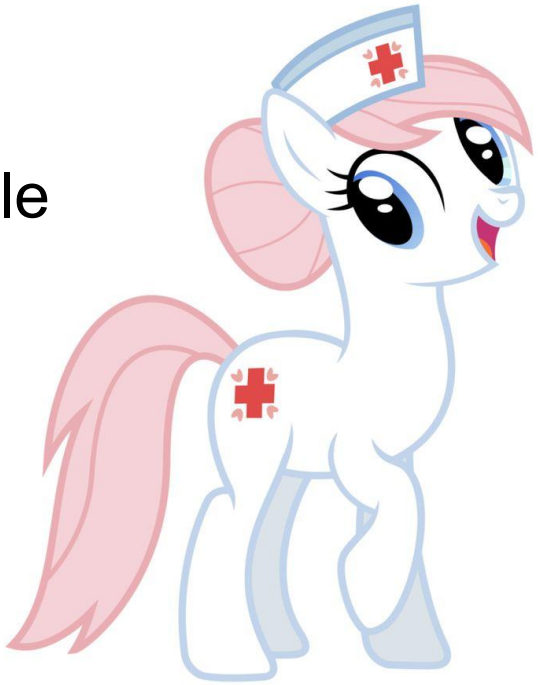
without visibility

the objectives of observability

- business velocity
- business availability
- business efficiency
- business scalability

the principles of observability

- store business and operations data together
- store at low resolution for core KPIs
- support self-service visualization
- keep your architecture simple and scalable
- democratize data for all
- collect and store once



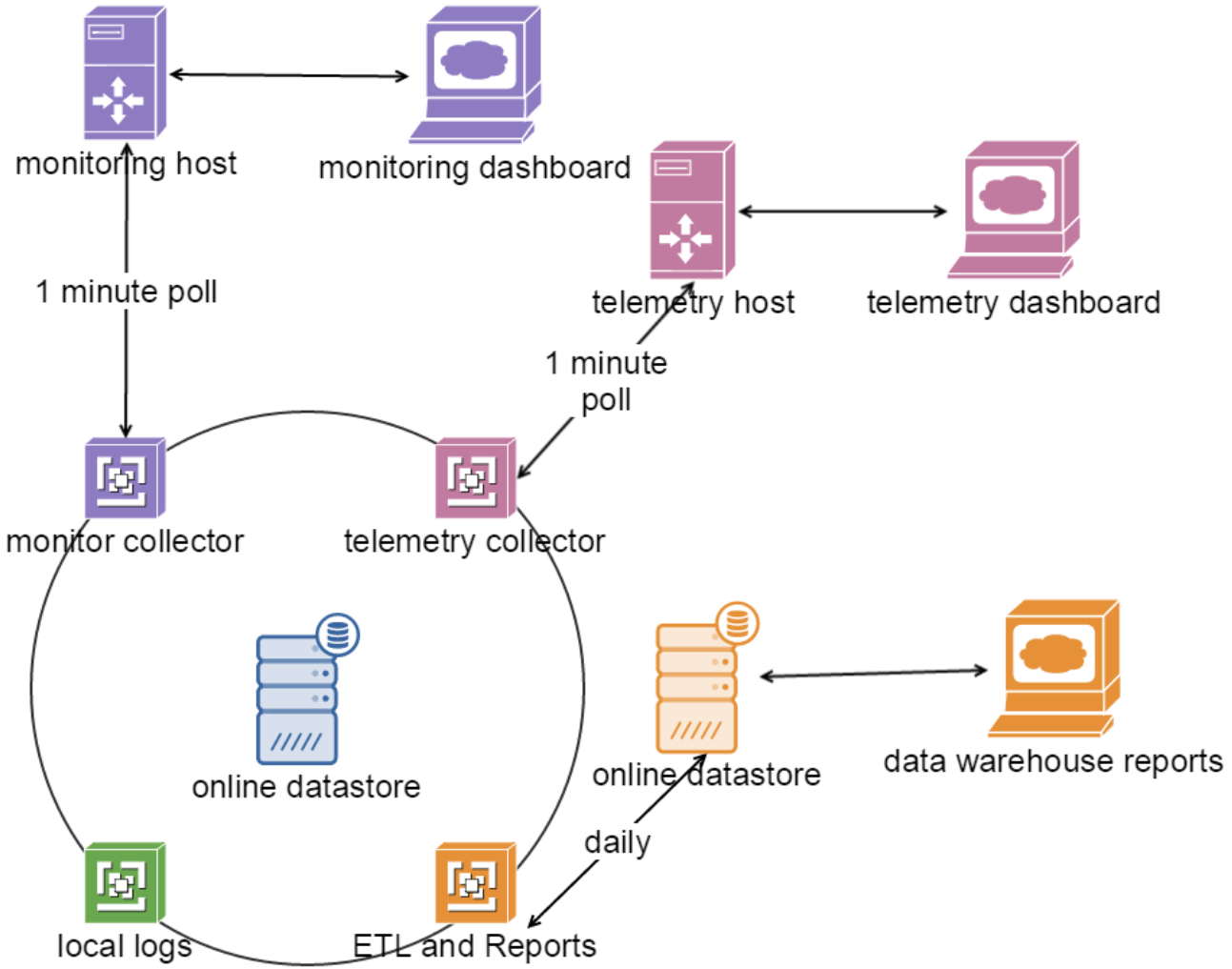
the outcomes of observability

- high trust and transparency
- continuous deployment
- engineering velocity
- happy staff (in all groups)
- cost-efficiency

state of the union

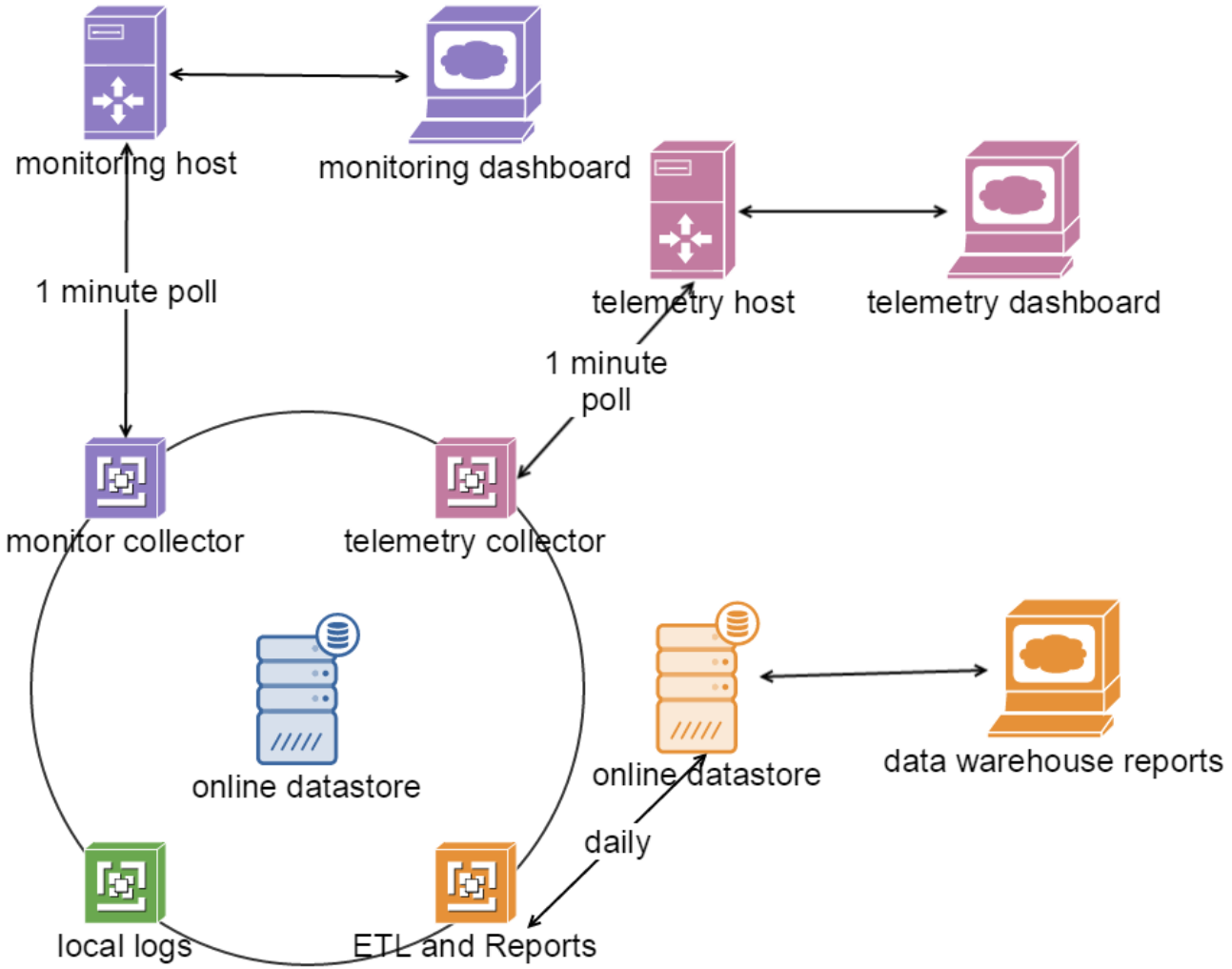
starting to address the traditional problems with opsviz

traditional monitoring



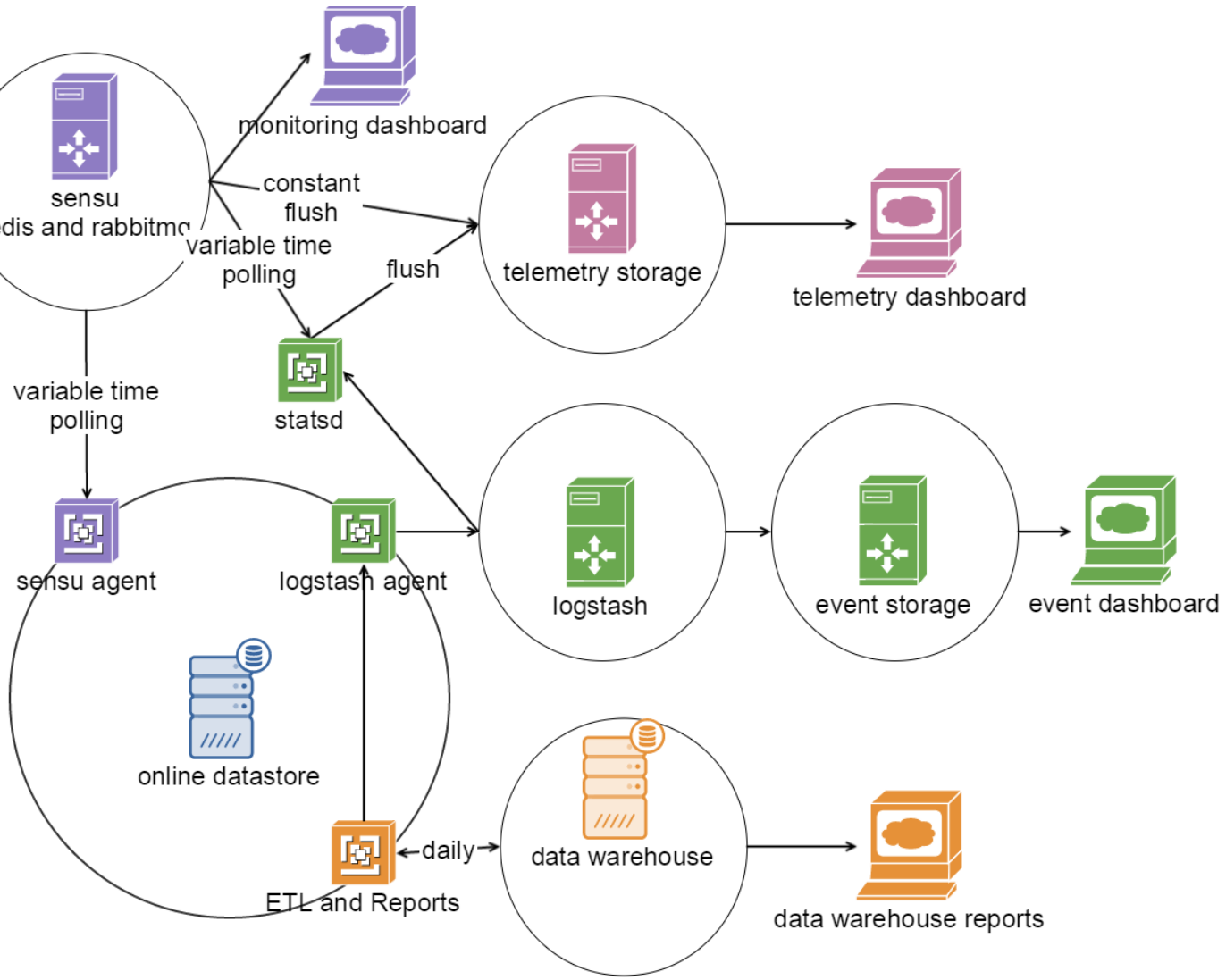
the problems

- too many dashboards
- data collected multiple times
- resolution much too high
- does not support ephemeral
- hard to automate
- logs not centralized



better...

- telemetry collected once
- logs centralized
- logs alerted on and graphed
- 1 second resolution possible
- supports ephemeral
- plays well with CM
- database table data into dashboards



Pythian
love your data

what must we improve?

- architectural component complexity and fragility
- functional automation and ephemeral support
- storage and summarization
- naive alerting and anomaly detection
- not understanding and using good math
- insufficient visualization and analysis

what's in a metric?

telemetry

- counters
- gauges

events

traditional

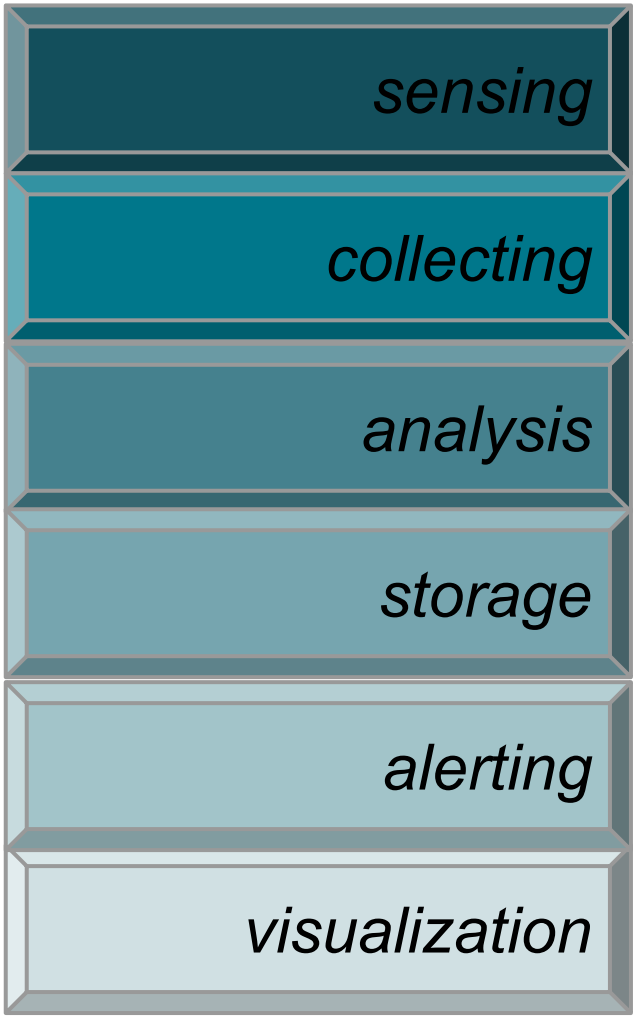
synthetic

resolution

latency

diversity

architectural components



architectural components

sensing

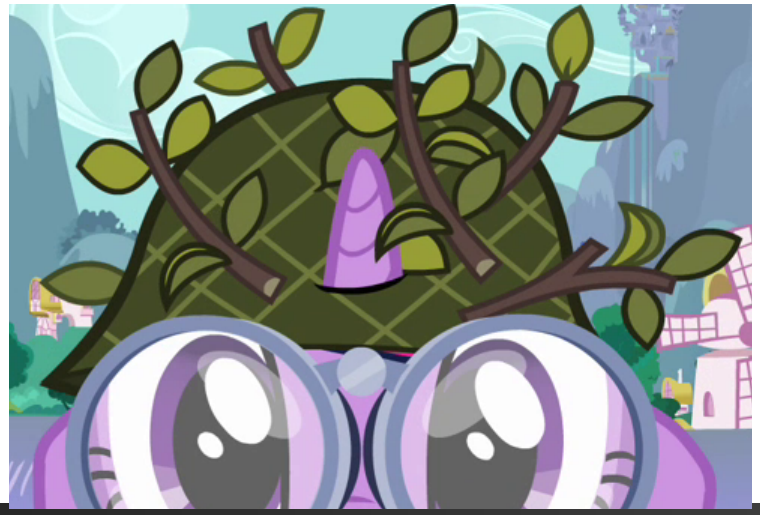
- Telemetry
- Events and Logs
- Applications
- Databases and SQL
- Servers and Resources

architectural components

- Agent or Agentless
- Push and Pull
- Filtering and Tokenizing
- Scaling
- Performance Impact

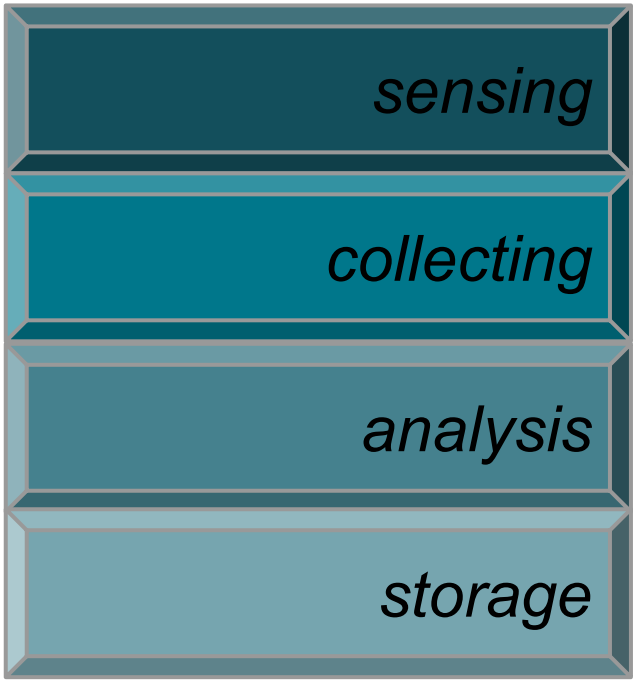
architectural components

- In-Stream
- Feeding into Automation
- Anomaly Detection
- Aggregation and Calculations



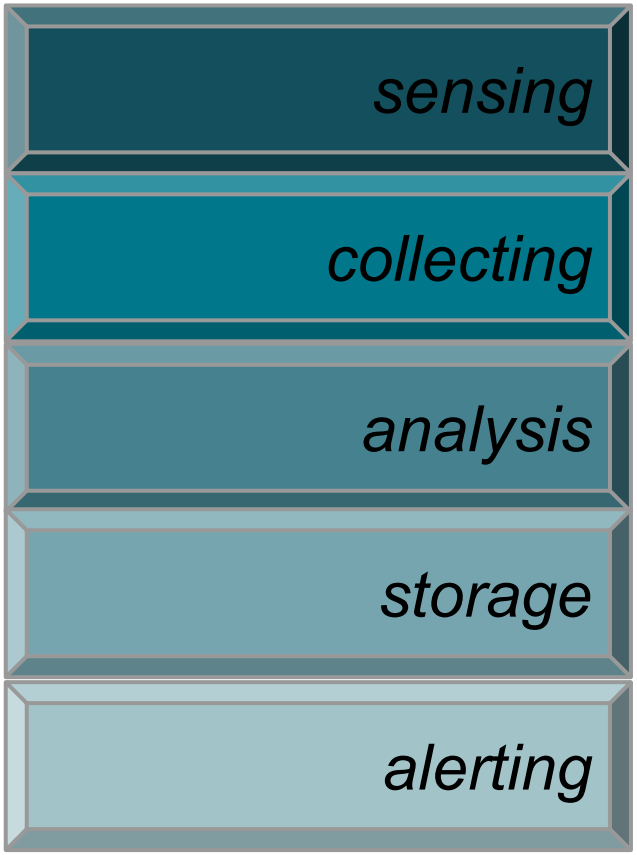
architectural components

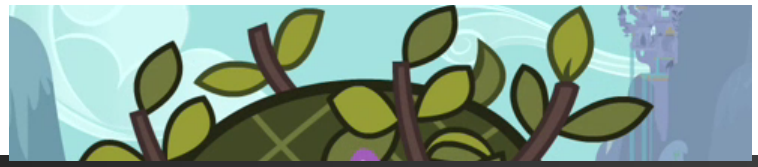
- Telemetry
- Events
- Resolution and Aggregation
- Backends



architectural components

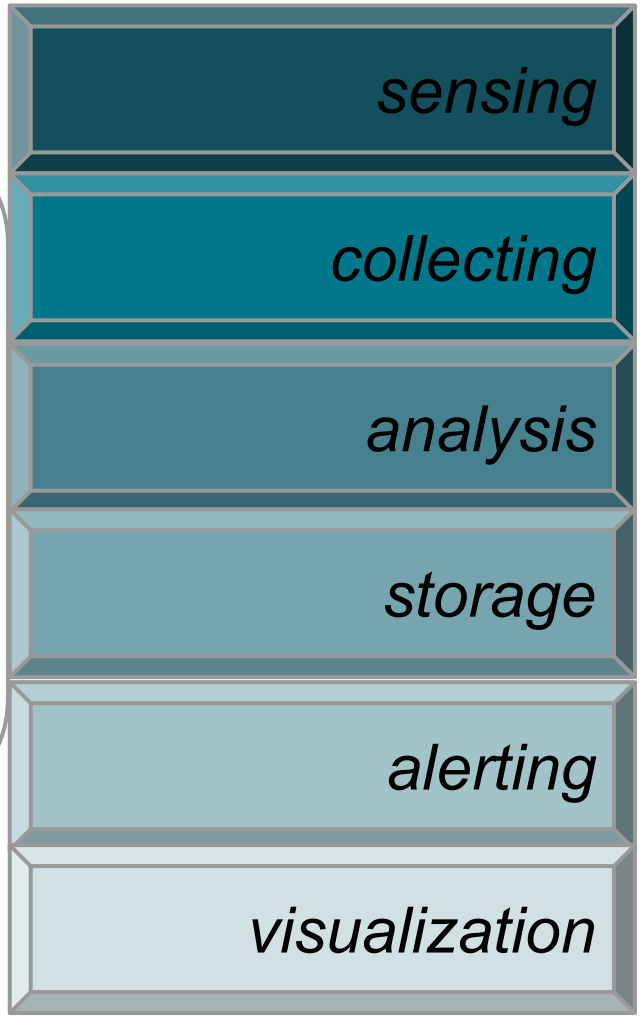
- rules-based processing
- notification routing
- event aggregation and management
- under, not over paging
- actionable alerts





architectural components

- executive dashboards
- operational dashboards
- anomaly identification
- capacity planning



what to measure?

we measure to support our KPIs

we measure to pre-empt incidents

we measure to diagnose problems

we alert when customers feel the pain

supporting our KPIs

- *velocity*
- *efficiency*
- *security*
- *performance*
- *availability*

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg

velocity

velocity

how fast can the org push new features?

how fast can the org pivot?

how fast can the org scale up or down?

deployment counts DB object changes data loads and changes	***provisioning counts*** cluster add/removal member add/removal	***engineering support*** query review turnaround data model review turnaround
deployment time DDL timings data load timings	***provisioning timing*** cluster add/removal member add/removal	
deployment errors failed DDL/DML schema mismatch	***provisioning errors*** cluster add/removal member add/removal	

Pythian
love your data

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg

efficiency

how cost-efficient is our environment?

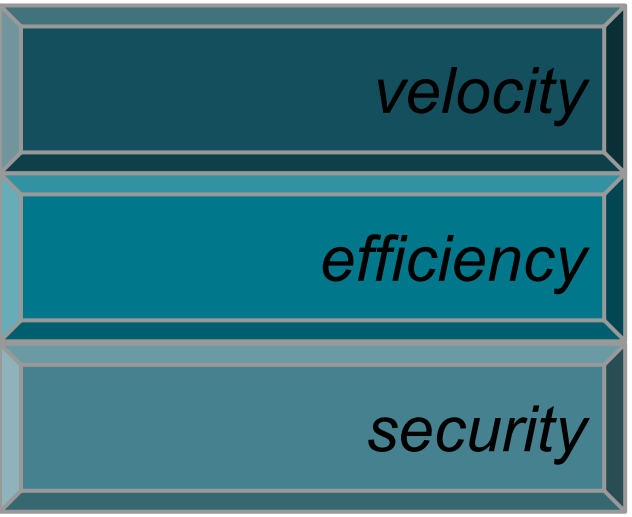
how elastic is our environment?

cloud spend data storage costs data compute costs data in/out costs	***physical spend*** data storage costs data compute costs data in/out costs	***staffing spend*** DBE spend Ops spend
cloud utilization database capacity database utilization	***physical utilization*** database capacity database utilization	***staffing utilization*** DBE utilization Ops utilization
provisioning counts cluster add/removal member add/removal	***application utilization*** percent capacity used mapped to product or feature	***staffing elasticity*** DBE/ops hiring time DBE/ops training time

Pythian
love your data®

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg

security

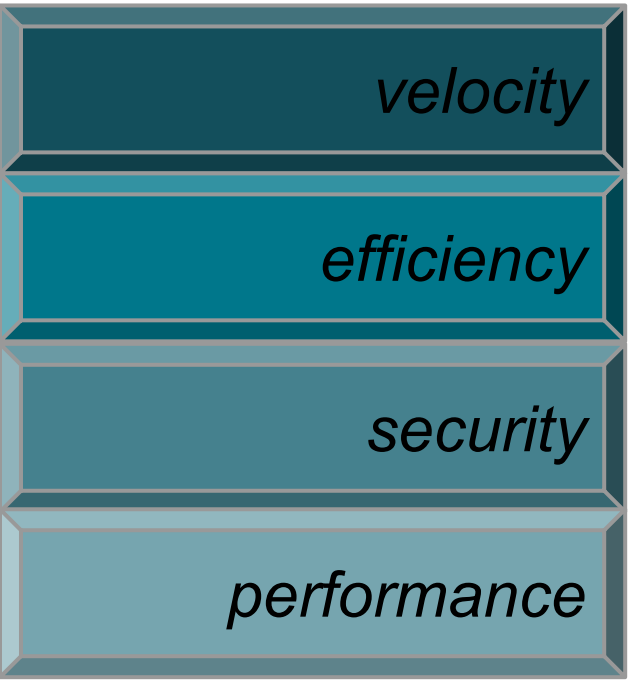


how secure is our environment?

penetration tests frequency success	***classified storage*** live in backups	***audit results*** frequency results
audit trail data utilization access	***users with access*** account access account audit	***infosec incidents*** event frequency

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg

performance



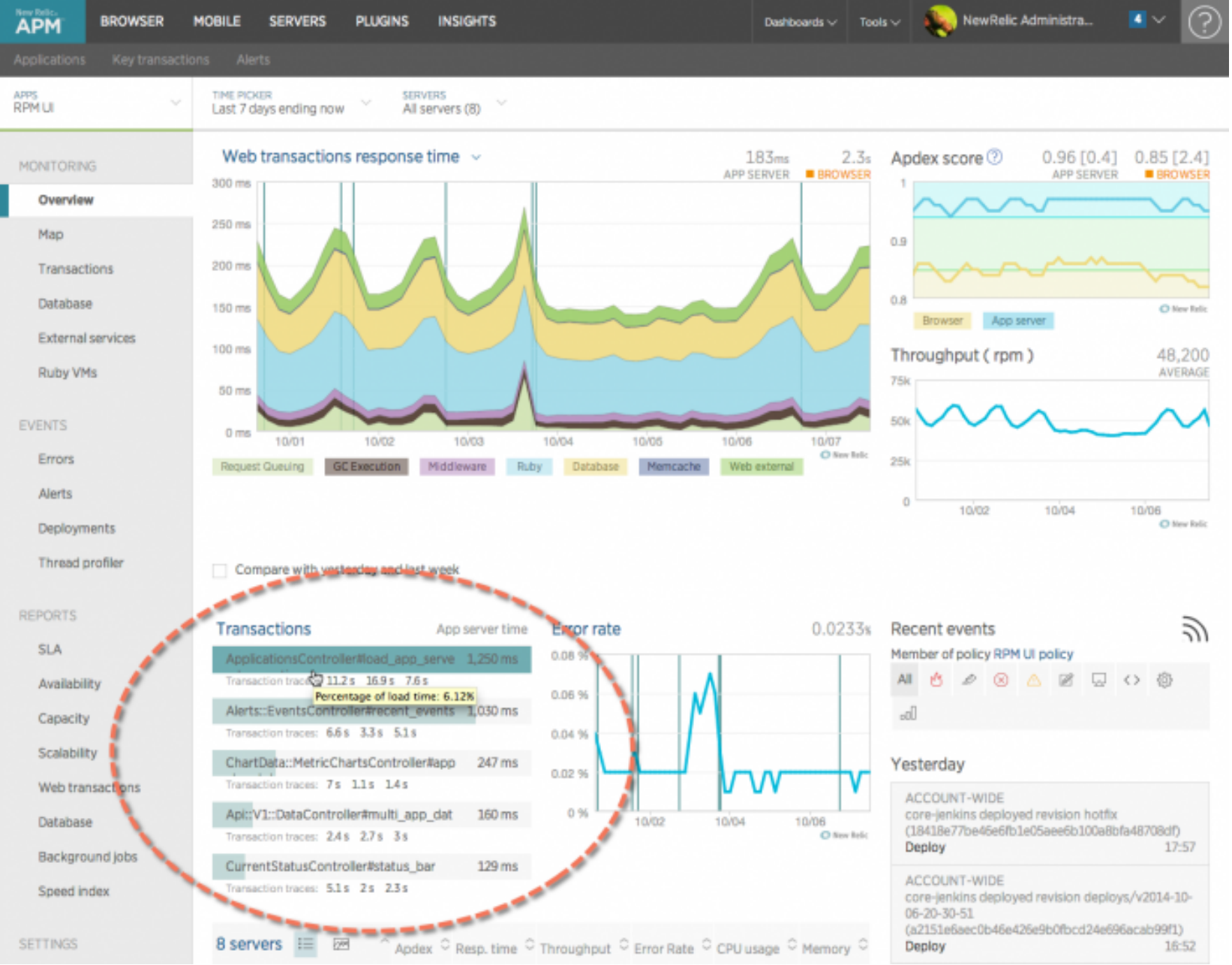
What is the AppDex of our environment?

AppDex(n), where n is the latency

AppDex(2.5), score of 95 indicates:

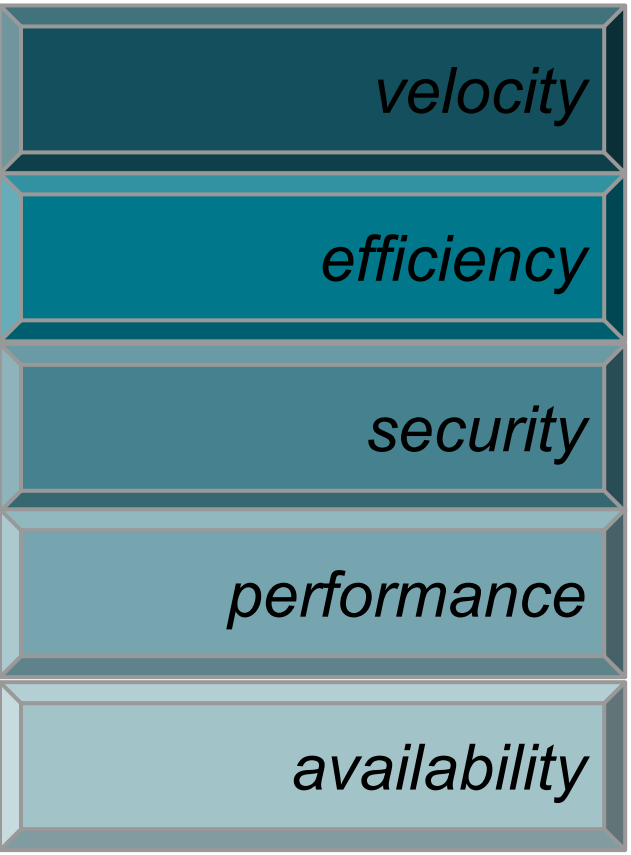
- 70% of queries under 2.5
- 25% of queries tolerable (5)
- 5% of queries as outliers

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg



Pythian
love your data

availability



how available is our environment to customers?

how available is each component to the application?

external response	***system availability***	***resource consumption***
pings	server uptime	CPU
websites	daemon uptime	storage
APIs	accessibility to app	memory
		network

https://derpicdn.net/img/view/2012/8/3/65841__safe_fluttershy_tank_scooter_artist-colon-giantmosquito_tortoise_vespa.jpg

pre-empting incidents supporting diagnostics

identify anomalies in latency or utilization

identify dangerous trends in latency or utilization

identify error rates indicating potential failure

measure as ***much*** as possible

alert on as ***little*** as possible

- align alerts to customer pain
- automate remediation if possible
- use metrics and events to solve what cannot be remediated

https://fc04.deviantart.net/fs71/i/2012/110/0/1/ninja_lyra_by_x72assassin-d4x0xqn.png

when measuring, understand...

your artificial bounds, which:

- ***to ensure resources are not exhausted***
- ***to ensure application concurrency stays in control***

your resource constraints

compromising on storage

telemetry data

collect and then flush

storing more than just averages

- min/max
- standard deviation
- percentiles for outlier removal

averages lie

storing histograms

the application

closest to perceived customer experience

documenting application components in SQL

understanding end to end transactions

prioritization by latency budgets

http://s3.amazonaws.com/bronibooru/941e30f3cb2129951df0d7d674fefcad.png

the application

application performance management tools

application logging to logstash

fire and forget to an event processing sys

telemetry for occurrence counts

histograms for visualizing

the server

the basics, resource utilization, process behavior and the network

log aggregation and measuring

- syslogs
- mysql logs
- cron, authentication, mail logs

aggregation up in distributed systems

the database: mysql

exposed database metrics

sql analytics and metrics

connection layer

how does they impact:

- availability KPIs
- concurrency KPIs
- latency KPIs

https://lockerdome.com/vox.com/7101308691941396

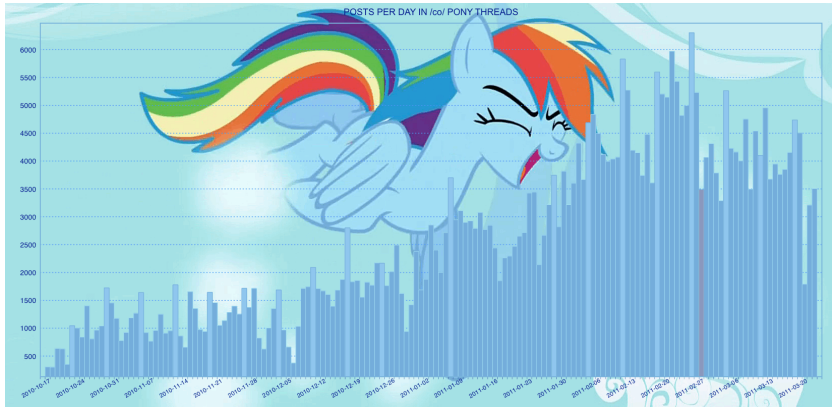
database metrics: workload

generic workload distribution

impacts to latency budgets

how fast are we hitting our resource and concurrency bounds?

- selects
- prepared statements
- ddl - data definition language
- dml - data manipulation language
- administrative commands



correlate DDL and admin to availability impacts

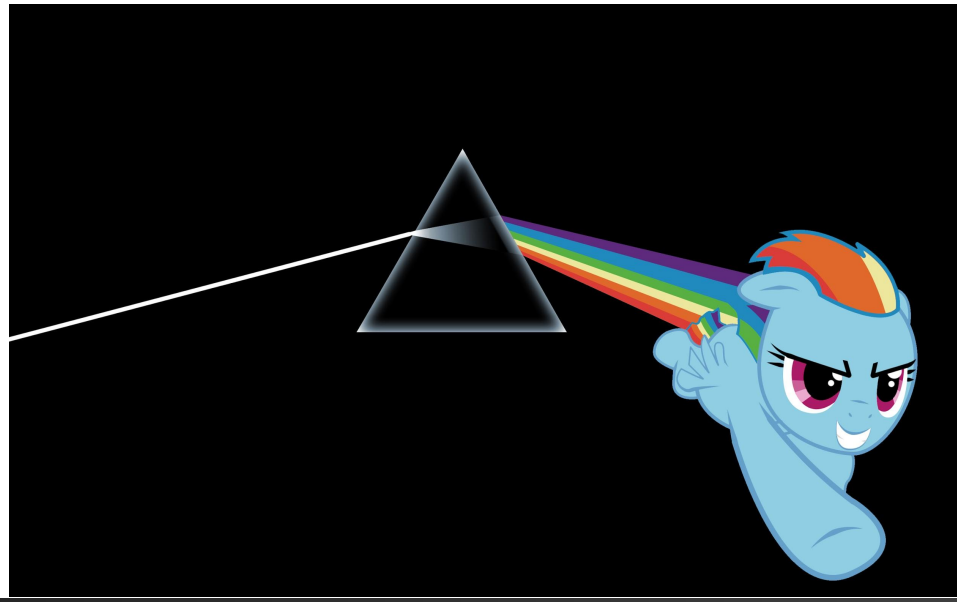
measure shifts in workload that may be impacting latency

database metrics: workload

data access behavior

- sort statistics
- join statistics
- handler status variables
  - index scans vs. full table scans
  - key index access
  - commits and rollbacks

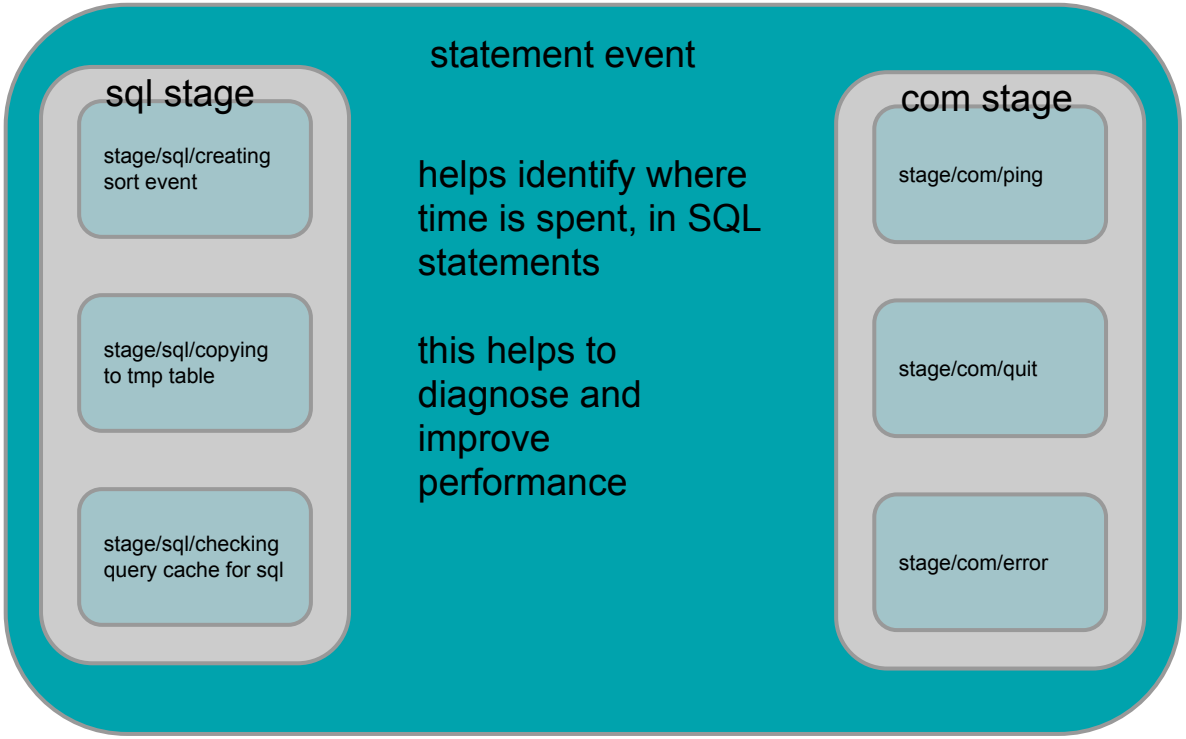
how are we impacting our latency budgets?



https://e621.net/data/e4/e4/e4e434d480672a67550c7c4113c85c73.jpg

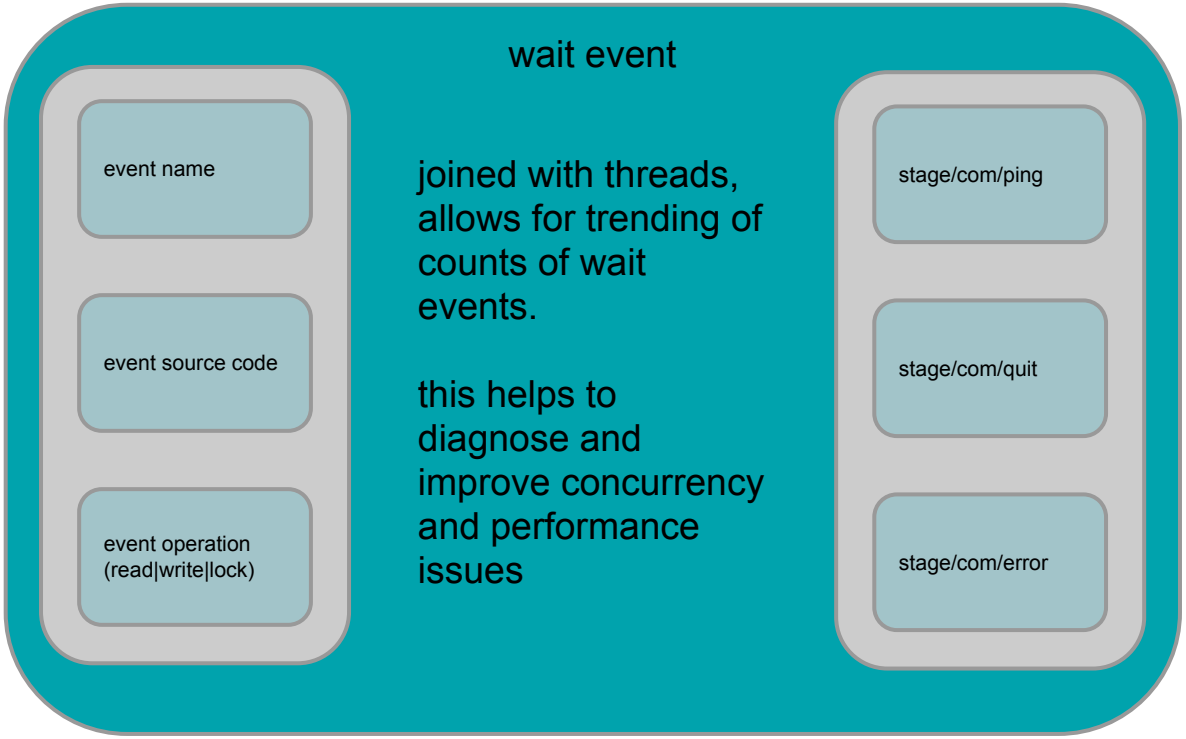
database metrics: workload

event metrics, inside out



database metrics: workload

where is time being spent?



database metrics: sql

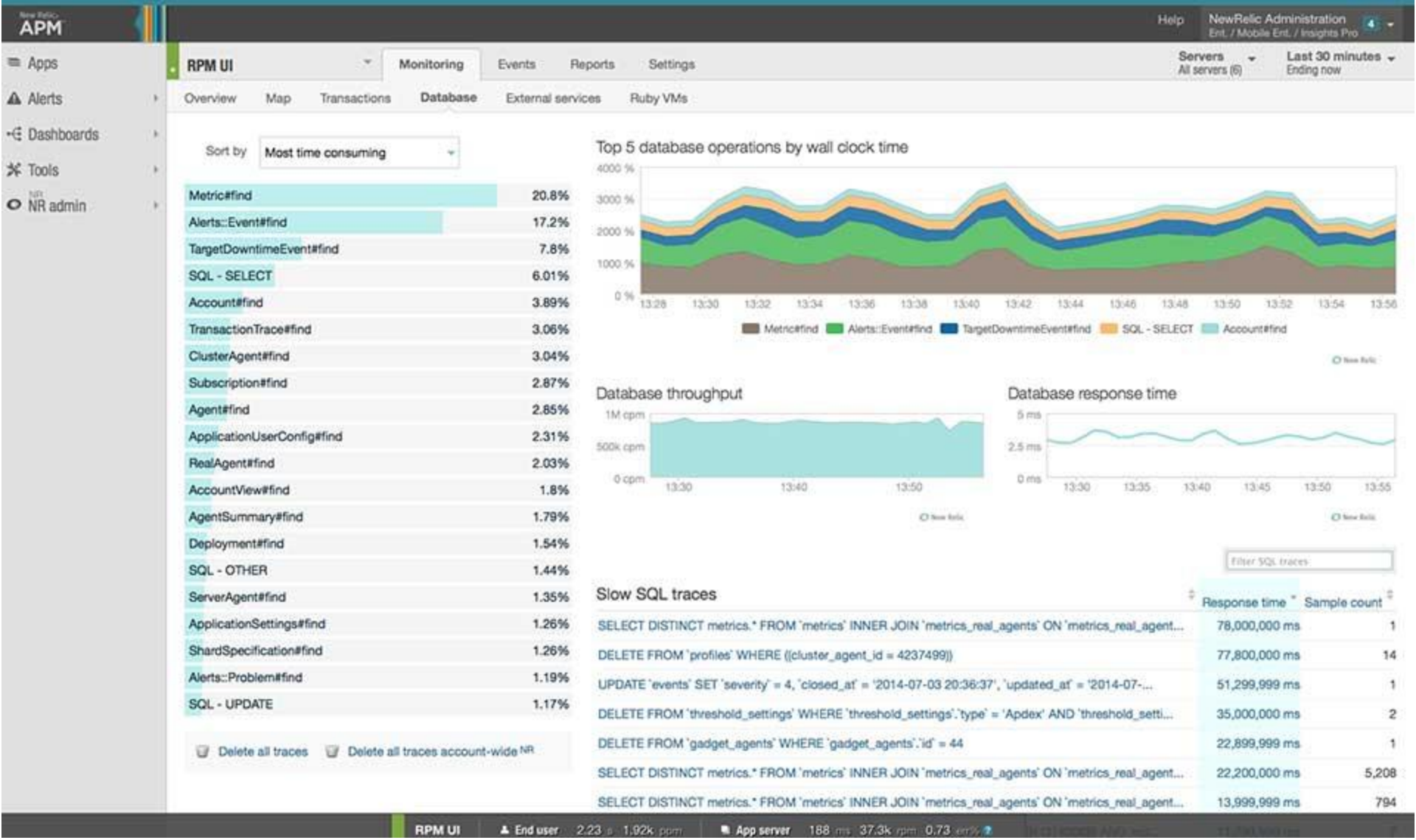
all sql should be logged, with ***context***

- comments pointing to application source code
- latency, and the components therein
- resources consumed
- data access paths taken

performance schema -> event and log analysis and visualization

slow logs -> event and log analysis and visualization

network sniffing -> event and log analysis and visualization



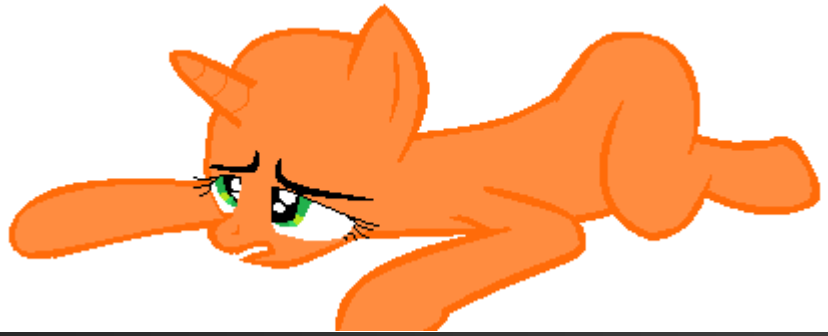
Pythian
love your data

database metrics: sql

all SQL should be logged, with ***context***

THIS IS HARD

solutions like vivid cortex can radically improve velocity



the database: connection layer

this is key to latency and availability KPIs

you must connect to your database

for latency, you have a fixed budget

getting to your network, other transaction components and SQL consume much of it

for availability, you must understand your bounds

tcp ports and mysql

max_connections (mysql)

- take one tcp port

time_wait (kernel)

- how long the port stays open
- 60 in most linux kernels
- effectively reduces port range by a factor of 60

port range of 30,000 limits to 5,000 total network connections

the database: network

mysql (5.6)

OS fixed amounts

- max tcp ports
- max tcp backlog
- network bandwidth

http://fc06.deviantart.net/fs71/f/2012/119/d/3/fat_pinkie_pie_by_nice123456-d4xy2w3.png

the database: network

mysql (5.6)

mysql configuration

- max_connections
- back_log
- max_connect_errors
- connect_timeout
- net_read_timeout
- net_write_timeout
- open_files_limit

connections: putting it together

use your network bounds
monitor proximity

status counters - network

- packets per sec
- request times
- requests per sec
- connection state (time_wait, listen, established)
- backlog
- socket queue drops and overflows
- time to get a tcp connection

the database: memory

operating system

shared memory, file descriptors, semaphores

- max shared memory segment size
- max number of segments
- max total of all memory available
- max file_descriptors per system and user
- max semaphores

the database: memory

mysql (5.6)

global memory bounds

- key_buffer_size
- innodb_buffer_pool_size
- innodb_additional_mem_pool_size
- innodb_log_buffer_size
- query_cache_size

the database: memory

mysql (5.6)

connection memory (max_connections)

- stack (thread_stack)
- connection and result buffers (net_buffer_length)
  - up to max_allowed_packet
- random read buffer (read_rnd_buffer_size)
- sequential read buffer (read_buffer_size)
- sort buffer (max of sort_buffer_size or max_heap_table_size)
- join buffer (join_buffer_size)
- (binlog_cache_size)

the database: memory

mysql (5.6)

max_connections = 1000

- thread_stack = 256k
- net_buffer_length x2 = 16k x 2 = 32k
  - up to max_allowed_packet x2 = 1m x2 = 2m
- read_rnd_buffer_size = 256k
- read_buffer_size = 128k
- sort_buffer_size = 256k max_heap_table_size = 16M
- join_buffer_size = 128k
- binlog_cache_size = 32k

total = 18.78m (reduce to ~3m by reducing max_heap…)
1000 connections = 2.9 GB

connections: putting it together

understand mysql impact to latency
understand proximity to mysql bounds

status counters: mysql

- processlist: connection and state counts
- thread statistics (thread_xxx)
- connection durations
- open_tables and open_files
- semaphores globally and per thread
- aborted_clients
- aborted_connects
- connection high water marks
- mysql network traffic
- mysql handlers - for buffer usage
- query response times

what's next?

better time series storage

- automatically distributed and federated, thus manageable and scalable
- leverage parallelism and data aggregation
- proper data consistency, backup and recovery
- instrumented and tuneable
- can consume billions of metrics and store them

what's next?

machine learning

- using code to pull out the signal from the noise
- easier correlation of metrics
- anomaly detection
- incident prediction
- capacity prediction
- REAL MATH

what's next?

consolidation

- business metrics
- telemetry data
- event and log text data

lab time

- set-up your hosts
- stretch, hydrate and use facilities
- 30 minutes

Pythian®
love your data

setup ec2

- https://github.com/dtest/plsc15-opvis

Pythian®
love your data

asbolus
the pythian opsviz stack

https://github.com/pythian/opsviz
http://dashboard.pythian.asbol.us/

resides in AWS, built via cloudformation and opsworks

internet facing rabbitMQ listener

- for external logstash/statsd/sensu clients
- using AMQP, SSL elastic load balancer
- in AWS VPC

asbolus
the pythian opsviz stack

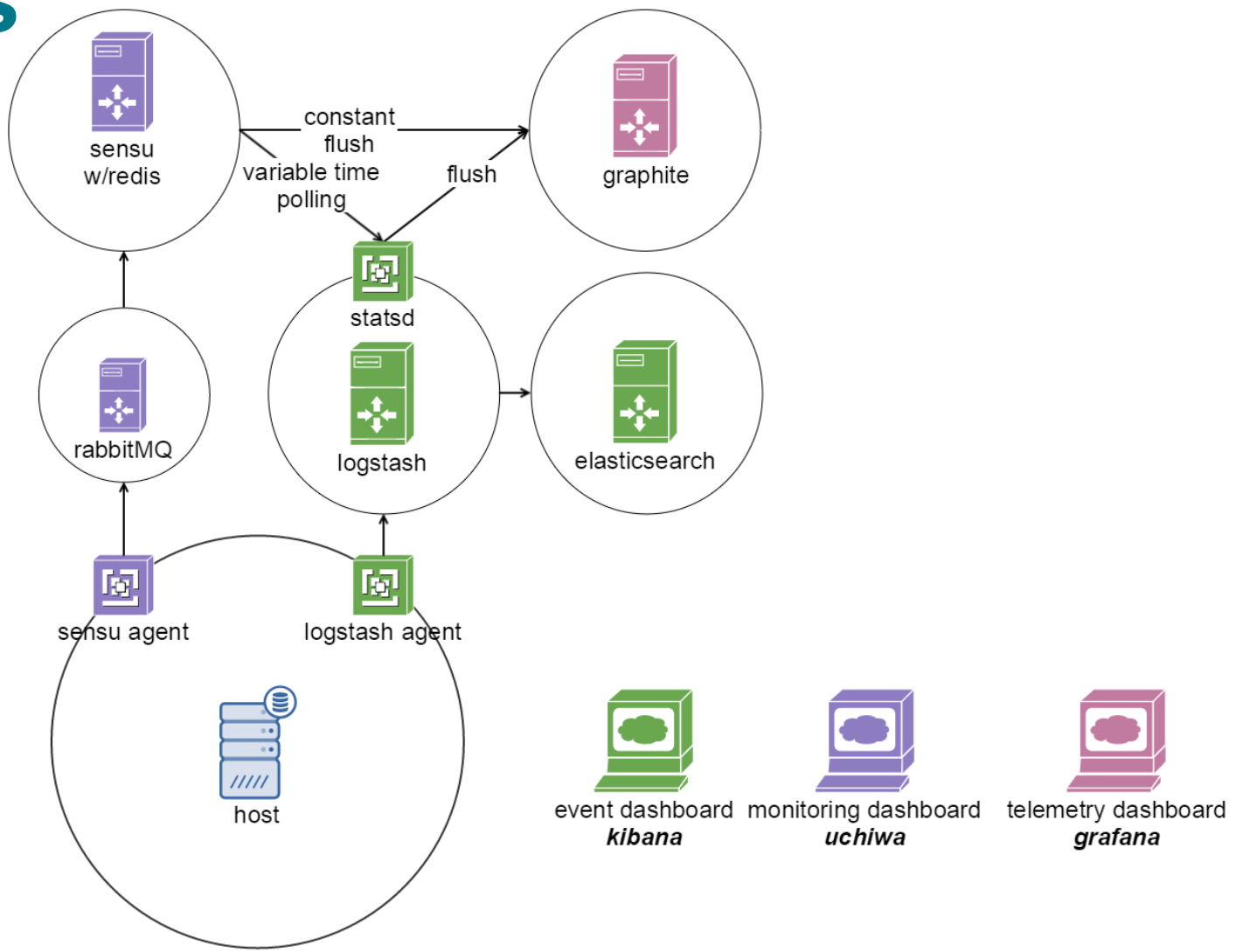
originally conceived and built by blackbird devops team

- taylor ludwig https://github.com/taylorludwig
- jonathan dietz https://github.com/jonathandietz
- aaron lee https://github.com/aaronmlee

continued development by pythian

- alex lovell-troy https://github.com/alexlovelltroy
- derek downey https://github.com/dtest
- laine campbell https://github.com/lainevcampbell
- dennis walker https://github.com/denniswalker

asbolus



Pythian
love your data

telemetry data: sensu

generated from sensu agent

- sensu agent on host polls from 1 to 60 seconds
- agent pushes to rabbitMQ
- rabbitMQ sends to sensu

once in sensu

- event handlers review
- flushed to carbon/graphite

telemetry data: logstash

generated from logstash agent

- logstash agent on host pushes to logstash server
- logstash server tokenizes and submits to statsD
- statsD flushes and sends to carbon/graphite

event data: logstash

generated from logstash agent

- logstash agent on host pushes to logstash server
- logstash server tokenizes and submits to elasticsearch

monitoring: sensu

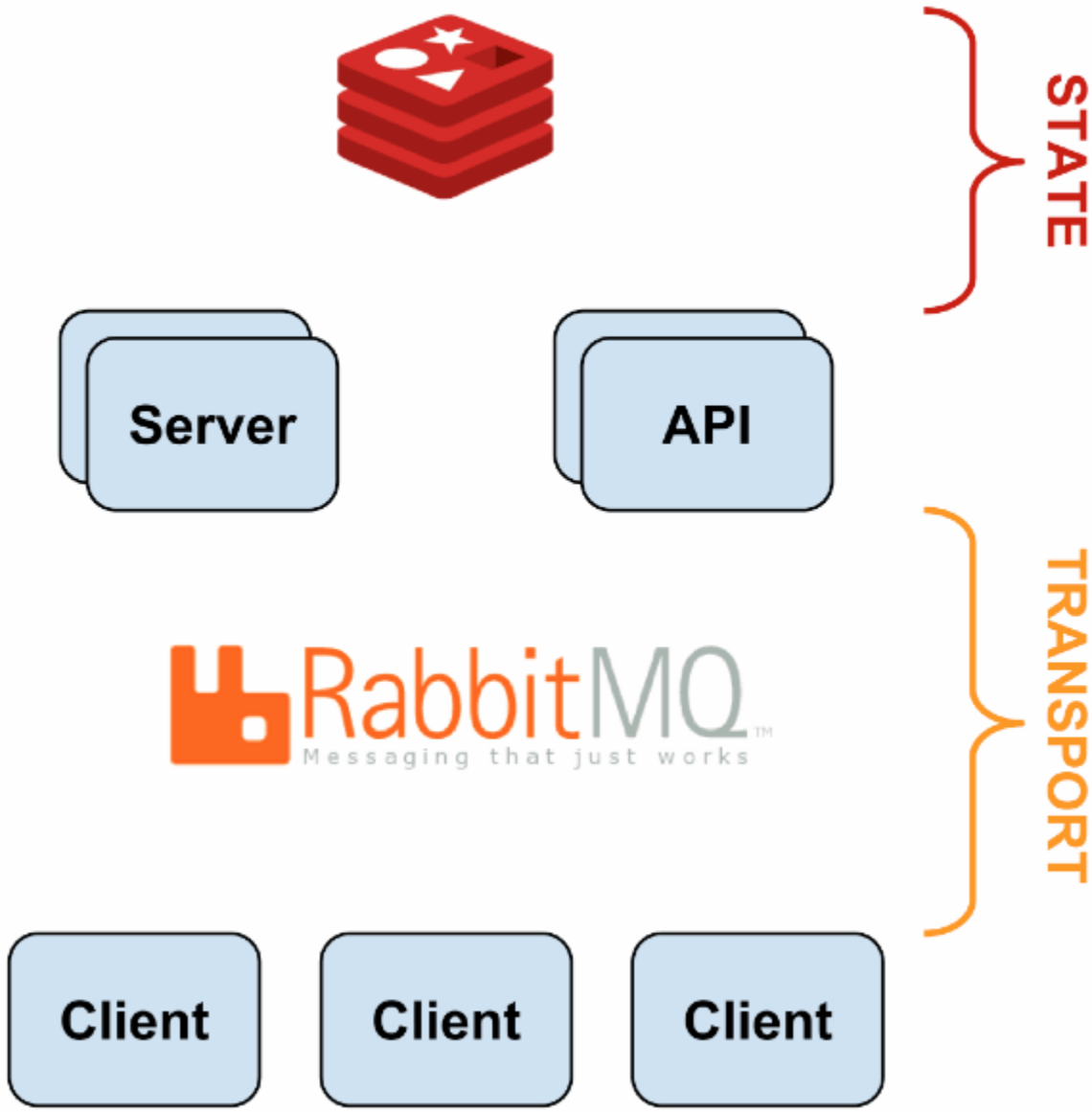
sensu server receives data from

- sensu agents
- statsD on logstash host

sensu handlers:

- flush to graphite
- send alerts to pagerduty
- create tickets in jira
- send messages to chat rooms

sensu architecture



monitoring: why sensu?

clients subscribe to checks, supporting ephemeral hosts

sensu server can be parallelized and ephemeral

clients easily added to configuration management

backwards compatible to nagios

multiple multi-site strategies available

excellent API

telemetry storage: graphite

why graphite?

- works with many different pollers, to graph everything!
- combines maturity with functionality better than others
- can be clustered for scale

what are the limitations?

- clustering for scale is complex, not native
- flat files means no joining multiple series for complex queries
- advanced statistical analysis not easy

event storage: elasticsearch

why?

- native distribution via clustering and sharding
- performant indexing and querying
- elasticity (unicorn scale)

what are the limitations?

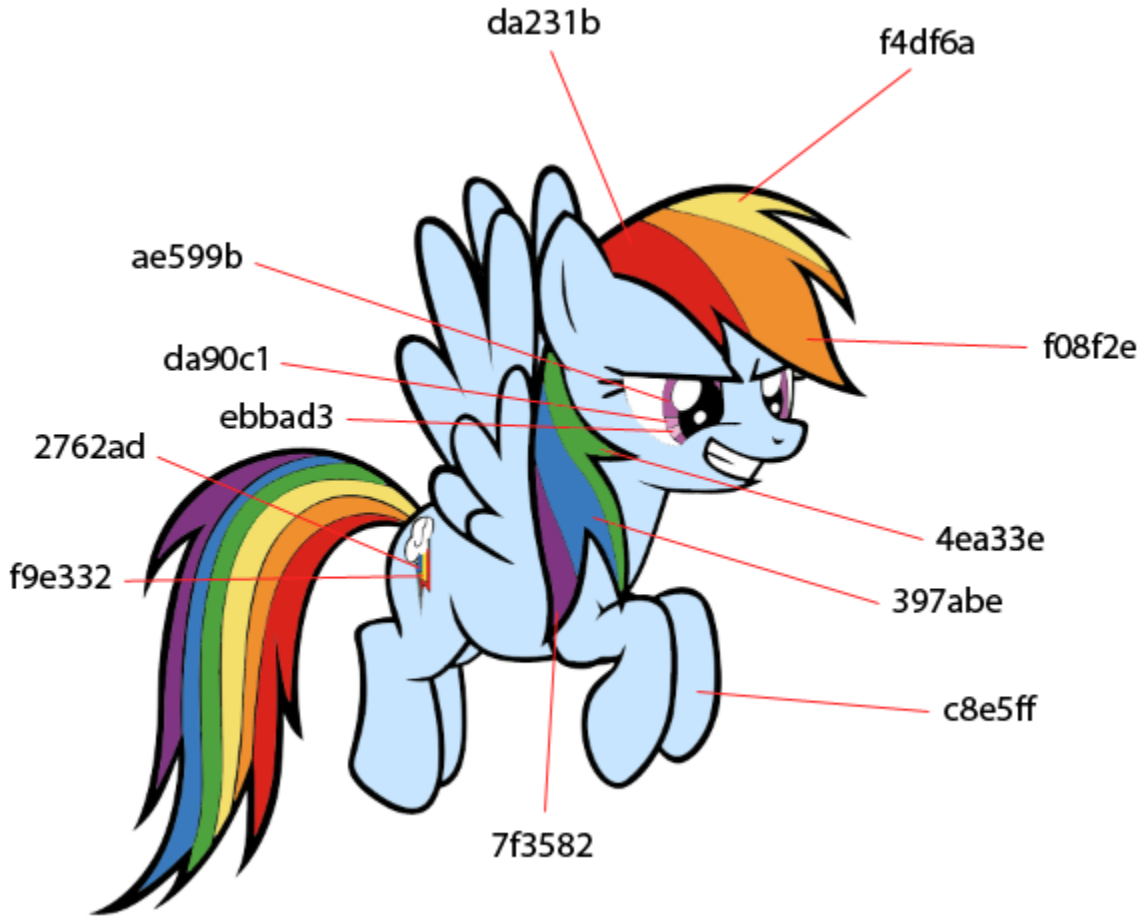
- security still minimal
- enterprise features becoming available (at a price)

visualization

telemetry: grafana

logs/events: kibana

incidents/alerts: uchiwa



scaled and available sensu

rabbitMQ

- network partition
  - pause-minority
- node failures
  - mirrored queues
  - tcp load balancers
- AZ failures
  - multiple availability zones
  - pause minority recovery
- scaling concerns
  - auto-scaling nodes
  - elastic load balancer

scaled and available sensu

sensu servers

- redis failure
  - elasticache, multi-AZ

- sensu main host
  - multiple hosts
  - use the same redis service
  - multi-AZ

monitoring your monitor

rabbitMQ

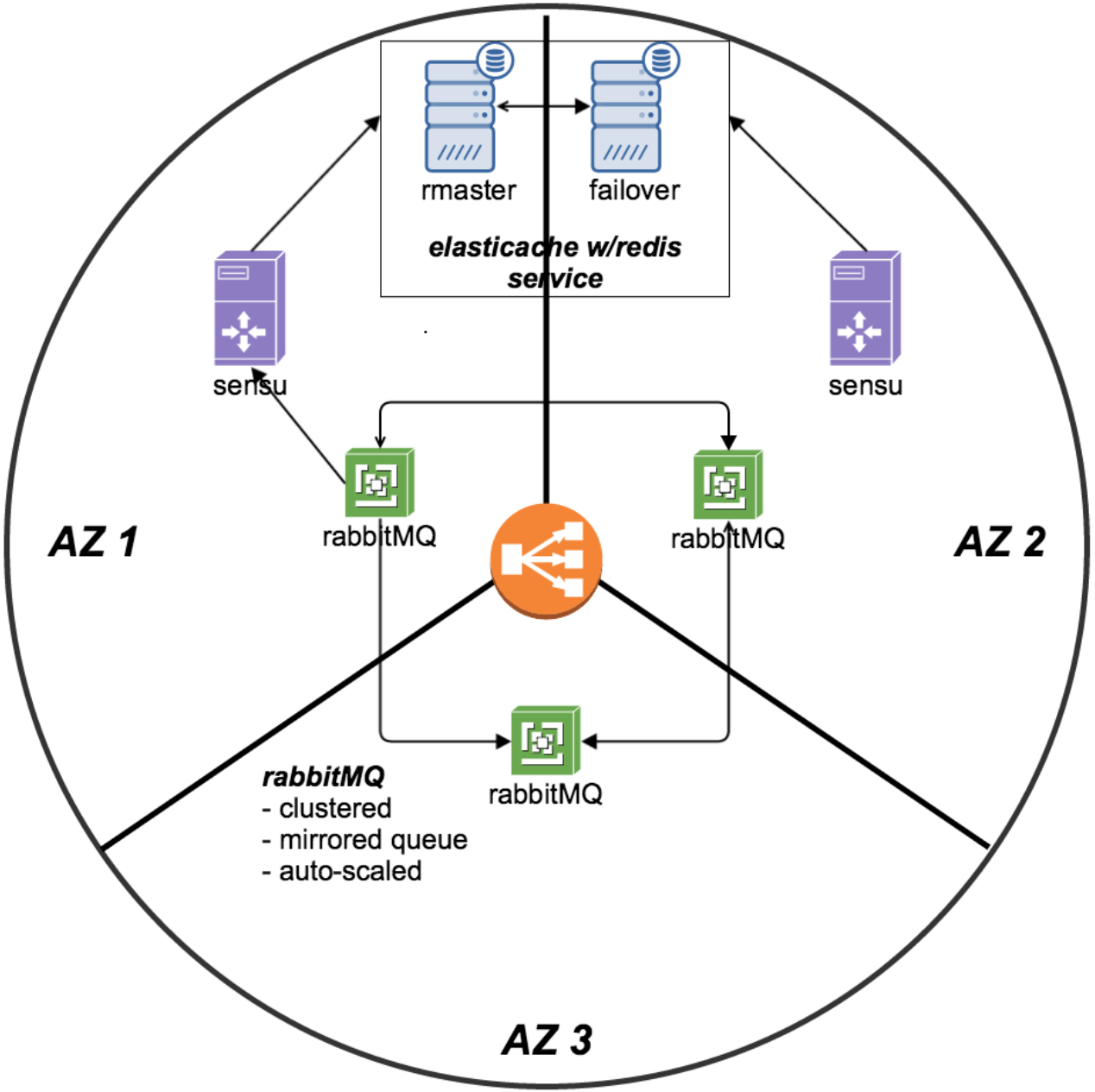
- monitor queue growth for anomalies
- monitor for network partitions
- monitor auto scaling cluster size

sensu

- sensu cluster size (n+1 sensu hosts)
- redis availability

workflow

- metric sent to ELB
- ELB sends to rabbitMQ cluster
- rabbitMQ
  - writes to master
  - replicates to mirror
- rabbitMQ sends to sensu



scaled and available graphite

carbon cache (tcp daemon, listening for metrics)

- scale with multiple caches on each host

carbon relay

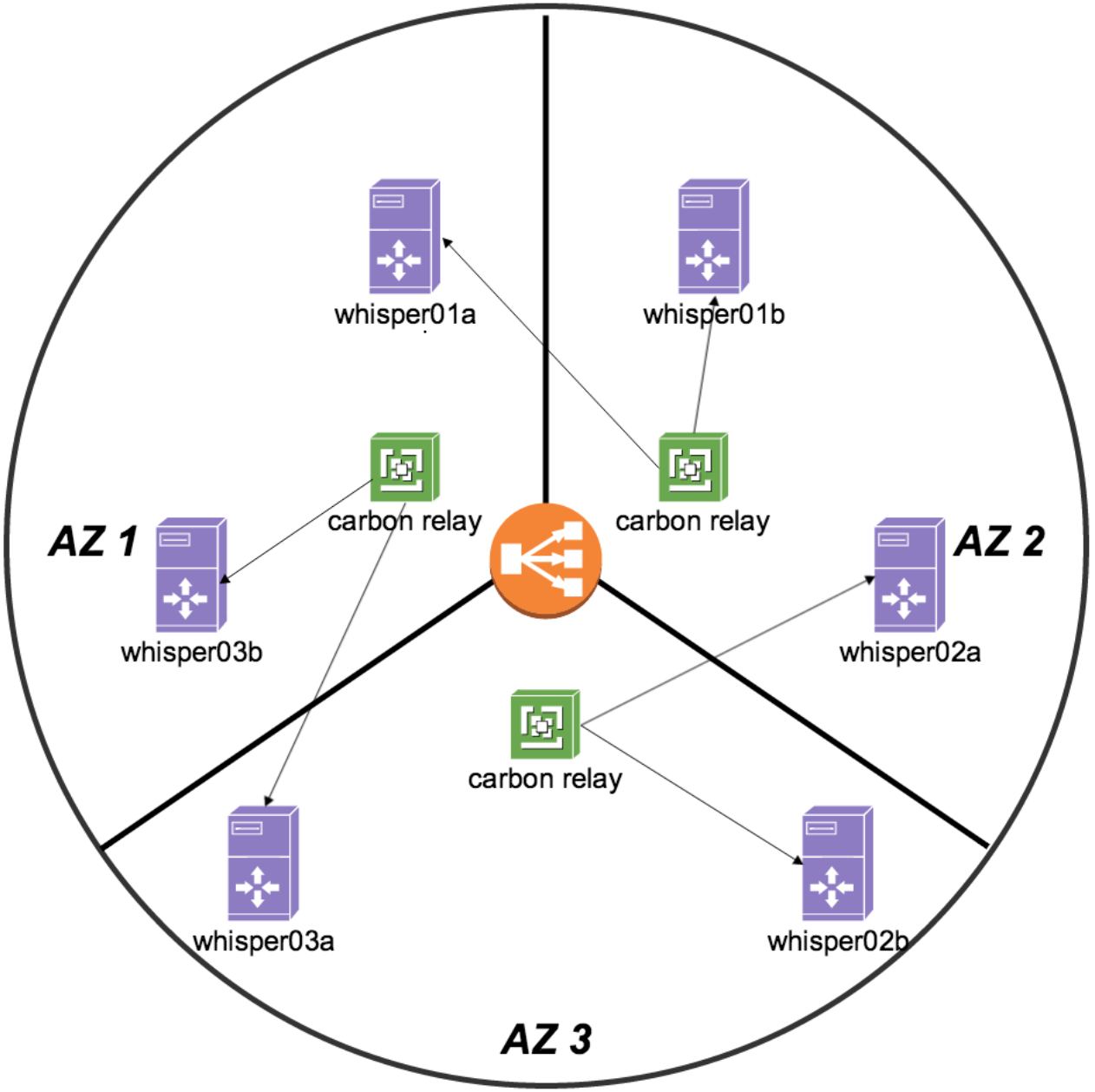
- used to distribute to multiple carbon caches
- by metric name, or consistent hashing
- can be redundant, using load balancers

whisper (flat file database, storage)

- can be replicated at the relay level
- running out of capacity and having to grow requires rehashing

workflow

- metric sent to ELB
- ELB sends to carbon relay
- carbon relay
  - chooses carbon cache
  - replicates as needed
- carbon cache flushes to whisper



scaling and available elasticsearch

node and cluster scaling

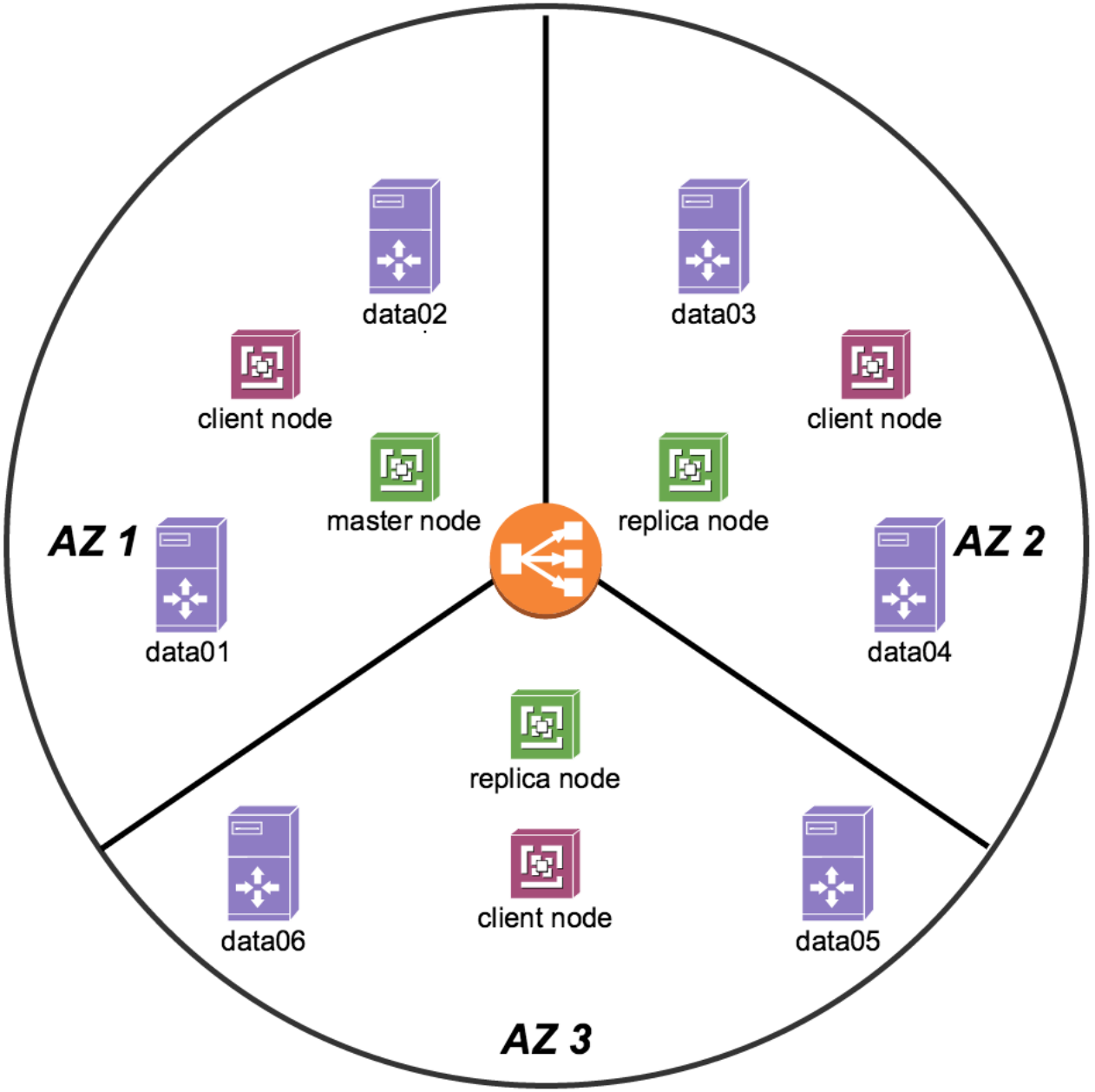
- clustering scales reads
- distribute across availability zones
- sharding indices allows for distributing data
- multiple clusters for multiple indexes

network partitions

- running masters on dedicated nodes
- running data nodes on dedicated nodes
- run search load balancers on dedicated nodes

workflow

- master/replica nodes route data and manage the cluster
- client nodes redirect queries
- data nodes store index shards



what's next?

visualization

- use sensu API to get incidents/alerts into graphite/
- merge kibana and grafana to one page

monitoring

- integrate flapjack for event aggregation and routing
- continue to add more metrics

full stack

- anomaly detection via heka or skyline
- influxdb for storage

lab time

- work on dashboards!
- 15 minute walkthrough w/ derek
- 45 minute play time

Pythian®
love your data

Q&A

email: downey@pythian.com
twitter: https://twitter.com/derek_downey

Pythian®
love your data